Bubble Cup 14 finals booklet

Table of contents

Problem A: Robot factory

Rising Stars only

Author:

Djordje Andjelkovic

Implementation and analysis:

Djordje Andjelkovic

Petar Vasiljevic

Statement:

You have received data from a Bubble bot. You know your task is to make factory facilities, but before you even start, you need to know how big the factory is and how many rooms it has. When you look at the data you see that you have the dimensions of the construction, which is in rectangle shape: $N\ x\ M$.

Then in the next N lines you have M numbers. These numbers represent factory tiles and they can go from 0 to 15. Each of these numbers should be looked in its binary form. Because from each number you know on which side the tile has walls. For example number 10 in its binary form is 1010, which means that it has a wall from the **North** side, it doesn't have a wall from the **East**, it has a wall on the **South** side and it doesn't have a wall on the **West** side. So it goes North, East, South, West.

It is guaranteed that the construction always has walls on its edges. The input will be correct.

Your task is to print the size of the rooms from biggest to smallest.

Constraints:

- $1 \leq N, M \leq 1000$

Input:

The first line has two numbers which are N and M, the size of the construction.

Next $N\ x\ M$ numbers represent each tile of construction

Output:

Once you finish processing the data your output consists of one line sorted from **biggest** to **smallest** room sizes.

Example input:

```
4 5
9 14 11 12 13
5 15 11 6 7
5 9 14 9 14
3 2 14 3 14
```

Example output:

```
9 4 4 2 1
```

Explanation:

For the input above, this is the look of the construction.

North means up. East is right, South is down, and West is left.

1001	1110	1011	1100	1101
0101	1111	1011	0110	0111
0101	1001	1110	1001	1110
0011	0010	1110	0011	1110

In the layout above we can see that we have one biggest room of 9 tiles, we have one room of 1 tile which is number 15 – 1111. On the most right we have one room of 2 tiles: 1101 and 0111, and two rooms of 4 tiles next to it. So, the output is: 9 4 4 2 1 sorted.

Time and memory limit: 1s / 256 MB

Solution and analysis:

It can be seen that given problem can be represented as graph where nodes are tiles and there is edge to the neighboring tile if there is no wall between them. Therefore, in order to calculate size of all of the rooms we have to find all connected components in graph and their corresponding sizes which can be done with simple DFS or BFS. After that, we have to sort sizes which can be done with counting sort so therefore overall complexity would be $O(NM)$.

Problem B: Array Game

Rising Stars only

Author:
Renea Mošo

Implementation and analysis:
Renea Mošo
Ivan Jevtić

Statement:

Alice and Bob are playing a game. They are given an array A of length N. The array consists of integers. They are building a sequence together. In the beginning, the sequence is empty. In one turn a player can remove a number from the left or right side of the array and append it to the sequence. The rule is that the sequence they are building must be strictly increasing. The winner is the player that makes the last move. Alice is playing first. Given the starting array, under the assumption that they both play optimally, who wins the game?

Input:

The first line contains one integer N - the length of the array A
The second line contains N integers $A_1, A_2, \ldots, A_N$

Output:

The first and only line of output consists of one string, the name of the winner. If Alice won, print "**Alice**", otherwise, print "**Bob**".

Constraints:

- $1 \leq N \leq 2 * 10^5$
- $0 \leq A_i \leq 10^9, 1 \leq i \leq N$

Example input 1:

5
1 2 3 2 3

Example output 1:

Alice

Explanation 1:

Alice can take the last element from the array. Now Bob cannot make a move, because the remaining options (1 and 2) will not make an increasing sequence.

Example input 2:

3
1 2 1

Example output 2:

Bob

Explanation 2:

Alice can number one from either side. Now Bob takes number 2, and Alice can't make a move.

Time and memory limit: 1s / 256 MB

Solution and analysis:

We have two cases:

- The player that is on the move can take a number from either the left or right end of the array.
- The player that is on the move can take a number only from one end of the array.

Let's first solve Case 2:

Since only the number on one end of the array can be taken, the other end of the array is forever blocked. That means that until the end of the game, the players can only take numbers from one end of the array. We can pre-calculate the length of the increasing sequence (to the left and to the right) for each number and based on the parity of the length of the increasing sequence we can determine who wins the game.

Case 1:

Let's say that the array is $A, C, \ldots., D, B$, and let's say that $A <= B$. If the player that is on the move takes B, that means that the left side of the array is now blocked, and then we are left with Case 2. If that happens, we already know the outcome. If the outcome is favourable for the player that is now on the move, he wins. Otherwise, he should take A, because then he might at least have a chance to win. Now we are left with the array $C, \ldots., D, B$, which we can recursively solve.

Time complexity: $O(n)$

Problem C: Party Organization

Rising Stars only

Author:

Nikola Pešić

Implementation and analysis:

Nikola Pešić

Renea Mošo

Statement:

On the great island of Baltia, there live N people, numbered from 1 to N. There are exactly M pairs of people that are friends with each other. The people of Baltia want to organize a successful party, but they have very strict rules on what a party is and when the party is successful. On the island of Baltia, a party is a gathering of exactly 5 people. The party is successful if either all the people at the party are friends with each other (so that they can all talk to each other, without having to worry about talking to someone they are not friends with) or no two people at the party are friends with each other (so that everyone can just be on their phones without anyone else bothering them).

Please help the people of Baltia organize a successful party or tell them that it's impossible to do so.

Input:

The first line contains two integer numbers, N and M – the number of people that live in Baltia, and the number of friendships.

The next M lines each contains two integers U_i and V_i – meaning that person U_i is friends with person V_i . Two friends cannot be in the list of friends twice (no pairs are repeated) and a person can be friends with themselves ($U_i \neq V_i$).

Output:

If it is possible to organize a successful party, print 5 numbers indicating which 5 people should be invited to the party.

If it is not possible to organize a successful party, print -1 instead.

If there are multiple successful parties possible, print any.

Constraints:

- $5 \leq N \leq 2 * 10^5$
- $0 \leq M \leq 2 * 10^5$
- $1 \leq U_i, V_i \leq \mathrm{N}$

Example input 1:

```
6 3
1 4
```

```
4 2
5 4
```

Example output 1:

```
1 2 3 5 6
```

Example input 2:

```
5 4
1 2
2 3
3 4
4 5
```

Example output 2:

```
-1
```

Time and memory limit: 1s / 256 MB

Solution and analysis:

If we have at least 48 people, we will always be able to find a successful party (because it is known that Ramsey's number for (5,5) is between 43 and 48 inclusive). If we have more than 48 people, we can just search for a successful party within the first 48 people, ignoring the rest. This means that we can solve the task with a brute-force algorithm. The complexity of this algorithm is $O(N^5)$.

Problem D: Bubble Strike

Premier League and Rising Stars

Author:

Nikola Smiljković

Implementation and analysis:

Nikola Smiljković

Uroš Berić

Statement:

Little Johnny Bubbles enjoys spending hours in front of his computer playing video games. His favourite game is Bubble Strike, a fast-paced bubble shooting online game for two players.

Each game is set in one of the *N* maps, each of them having different terrain configurations. First phase of each game decides on which map the game will be played. The game system randomly selects three maps and shows them to the players. Each player must pick one of those three maps to be discarded. The game system then randomly selects one of the maps that were not picked by any of the players and starts the game.

Johnny is deeply enthusiastic about the game and wants to spend some time studying maps, thus increasing chances to win games played on those maps. However, he also needs to do his homework, so he does not have time to study all the maps. That is why he asked himself the following question: "What is the minimum number of maps I have to study, so that the probability to play one of those maps is at least *P*"?

Can you help Johnny find the answer for this question? You can assume Johnny's opponents do not know him, and they will randomly pick maps.

Input:

The first line contains two integers *N* and *P* - total number of maps in the game and probability to play map Johnny has studied.

Output:

Output contains one integer number - minimum number of maps Johnny must study.

Constraints:

- $3 \leq N \leq 1000$
- $0 \leq P \leq 1$

Example input 1:

```
7 1
```

Example output:

```
6
```

Time and memory limit: 0.5s / 256 MB

Solution and Analysis:

Let us determine what is the probability for a map Johnny has studied to be chosen for the next game out of three randomly selected maps.

We know that Johnny smartly discards maps, so he will never discard a map he has studied unless he has studied all three of them. Also, the other player does not know which maps Johnny has studied, so he will not always target them.

With three randomly selected maps by the game system there are four cases: Johnny has studied exactly three, two, one or none of those maps.

If Johnny has studied at least two of them, Johnny can discard the map he has not studied (if such exists) and the game will be played on a map he has studied. If he has studied exactly one map of the three, then the probability is 50%, depending on the opponent's choice. Lastly, if he has not studied any of the maps, the probability is 0%.

Thus, the formula which determines chances for a map Johnny has studied to be the chosen is:

$$P = \frac{(C_3 + C_2) * 100\% + C_1 * 50\% + C_0 * 0\%}{C_3 + C_2 + C_1 + C_0}$$

where C_i represents the number of ways the system can select three maps out of N and Little Johnny Bubbles has studied exactly *i* maps out of those three maps.

Now, let us determine values C_i depending on N, the total number of maps, and K, the number of maps Little Johnny Bubbles has studied.

Since there are K maps Johnny has studied and $N - K$ maps he has not studied, we have that: $C_3 = \binom{K}{3} * \binom{N-K}{0}, C_2 = \binom{K}{2} * \binom{N-K}{1}, C_1 = \binom{K}{1} * \binom{N-K}{2}$ and $C_0 = \binom{K}{0} * \binom{N-K}{3}$.

At the end, it is left to find value K. We can try each value for K up to N, and pick the smallest value where probability is larger or equal to the target probability.

Additionally, it can be proven that larger K has larger probability so binary search by K can be performed for more optimal solution. However, considering problem constraints, it is not necessary.

Problem E: Hidden Fortress

Premier League and Rising Stars

Author:

Nikola Pešić

Renea Mošo

Implementation and analysis:

Nikola Pešić

Aleksandar Cvetić

Statement:

As part of your contribution in the Great Bubble War, you have been tasked with finding the newly built enemy fortress. The world you live in is a giant 10^9 x 10^9 matrix, with coordinates of the matrix squares numbered from 1 to 10^9.

You know that the enemy base has the shape of a rectangle, with the sides parallel to the sides of the matrix. The people of your world are extremely scared of being at the edge of the world, so you know that the base does not contain any of the squares on the edges of the matrix (the x or y coordinate being 1 or 10^9).

To help you locate the base, you have been given a device that you can place in any square of the matrix, and it will tell you the Manhattan distance to the closest square of the base. The Manhattan distance from square (a, b) to square (p, q) is calculated as $|a - p| + |b - q|$. If you try to place the device inside the enemy base, you will be captured by the enemy. Because of this, **you need to make sure to never place the device inside the enemy base**.

Unfortunately, the device is powered by a battery, and you cannot recharge it. This means that you can use the device **at most 40 times**.

Input:

This problem is interactive.

The input contains the answers to your queries.

Output:

Once you are sure where the enemy base is located, you should print "! x y p q," where (x, y) is the square inside the enemy base with the smallest x and y coordinates, and (p, q) is the square inside the enemy base with the largest x and y coordinates.

Interaction:

Your code is allowed to place the device on any square in the matrix by writing "? i j." In return, it will receive the Manhattan distance to the closest square of the enemy base or -1 if the square you placed the device on is inside the enemy base or outside the matrix.

Your solution should use no more than 40 queries.

This is an interactive problem. You must use a flush operation right after printing each line. For example, in C++ you should use the function fflush(stdout), *in Java* — System.out.flush(), *in Pascal* — flush(output) *and in Python* — sys.stdout.flush().

Time and memory limit: 1s / 256 MB

Solution and Analysis:

Problem can be solved using just 4 queries. We will use the first two queries to find a point on the middle of one side of the fortress by querying Manhatton distance for $d_1 = query(1, 10^9)$ and $d_2 = query(10^9, 10^9)$. By getting these two distances we can now calculate y_m coordinate of middle of one side of the fortress. $y_m = 1 + \frac{10^9 - 1 - d_1 + d_2}{2}$

Then by querying $d_3 = query(10^9, y_m)$ we can get distance from one side of the fortress and from querying $d_4 = query(1, y_m)$ we can get distance from other side.

Then we can calculate coordinates of all four corner points of the fortress:

$$x = 1 + d_4$$

$$y = 1 + |d_2 - d_3|$$

$$p = 10^9 - d_3$$

$$q = 10^9 - |d_1 - d_3|$$

Problem F: Restaurant Game

Premier League and Rising Stars

Author:

Aleksandar Damjanović

Implementation and analysis:

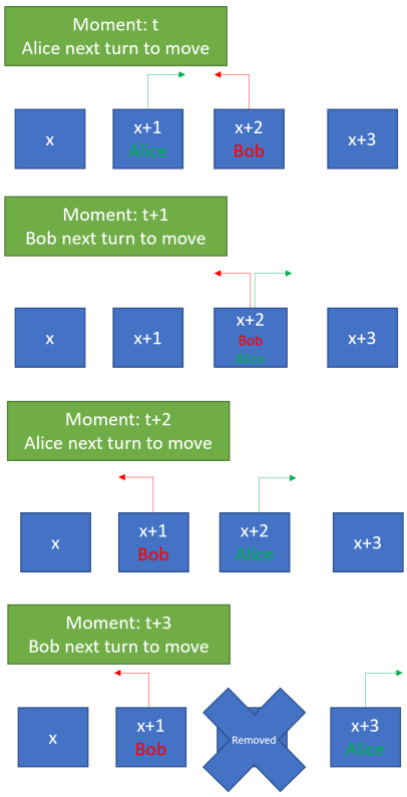
Aleksandar Damjanović

Uroš Berić

Statement:

Alice and Bob always had hard time choosing restaurant for the dinner. Previously they performed Eenie Meenie Miney Mo game, but eventually as their restaurant list grew, they had to create a new game.

This new game starts as they write restaurant names on N cards and align the cards in one line. Before the game begins, they both choose starting card and starting direction they are going to. They take turns in order one after another. After each turn, they move one card in their current direction. If they reach the end or beginning of the line of cards they change direction. Once they meet in a card, the card is marked for removal and is removed the first moment they both leave the card.



They repeat this process until there is only one restaurant card left. Since there are a lot of restaurant cards, they are bored to simulate this process over and over and need your help to determine the last card that remains. Can you help them?

Input:

The first line of the input is one integer T representing number of test cases.

Each test case contains 3 lines:

The first line contains an integer N representing initial number of cards.

Next line contains two integer values A, B representing starting 0-based index of the card in the array.

Last line contains two strings D_A, D_B representing starting direction of their movement. Direction is represented with lower case English letters and can either be "left" or "right".

Output:

The output contains T integer number – the 0-based index of the last card that remains for every test case in order.

Constraints:

- $1 \leq T \leq 10^4$
- $2 \leq N \leq 10^{18}$
- $0 \leq A, B < N$
- $A \neq B$
- $D_A, D_B \in \{left, right\}$

Example input:

```
1
4
0 1
left right
```

Example output:

```
0
```

Explanation:

Note that since Alice is starting at the beginning of the line even though her initial direction is left, on her next move she will go right.

Time and memory limit: 1s / 128 MB

Solution and analysis:

Firstly, let's narrow down possibilities for the solution by proving the following statement:

The last node that remains is either node with index $\mathbf{0}$ *or node with index* $N-1$.

Let's assume that node **0** is about to be removed. We are then in a situation as following: **ABOO…OO**, where $\boldsymbol{A}$ is Alice standing on node **0**, $\boldsymbol{B}$ is Bob standing on one node to the right, **O** are the rest of the nodes and it's the Bobs move to go left. After that they both will be moving in the right direction removing nodes one by one until the far-right node. Same applies when node $\boldsymbol{N-1}$ is about to be removed. We conclude that when a node at the very end of the list needs to be deleted, all nodes are deleted all the way to the opposite end.

So, if node **0** is deleted before node $\boldsymbol{N-1}$, node $\boldsymbol{N-1}$ is the last to stand and vice versa.

Thus, *the solution is either node 0 or node* $N-1$.

Let's go to the situation in which directions of Alice and Bob are equal. If initial directions are opposite, then we need to calculate which player will come to the edge first where he will change the direction.

Let's assume their initial directions are opposite. If Alice is left to Bob and he moves to the right, it means that the two players will meet at some point, so one node will get deleted in the process. Afterwards, once we determine which player will come to the edge first and the number of moves it takes him to do that, we will then calculate their new positions and distance between them, also paying attention if they have met in which case, we will just decrease the distance by one.

Now, they are moving in the same direction. Note that when the players meet and a node gets deleted, the next time they have the same direction the distance between them will be less than before. Every time after (or before, if you wish) they meet there will be a situation in which they have the same direction, because they move alternately one player will always come to the edge first and change his direction. That means the distance between them decreases by one every time they meet, and in between their meetups the direction changes, alternately. When we find the first position in which they have the same direction, we can determine after how many meetups and changes of the direction will there be no nodes between them (the distance will become 0) and they will be deleting nodes one by one in the direction they are moving.

We have found the position in which they have the same direction and the distance between them. They moved the same number of steps after their initial position, so the next to move is Alice.

Let us say we are in the following situation:

…B000A… and they both move left. The distance between them is 4 and they move left, after next meetup it will be 3 and the direction will be right etc. At the end the distance between them will be 1 (***…AB…***) and they will be moving right. Here the players will just start removing nodes one by one to the far right, thus the last one standing will be node 0. (Note that they can't remove node 0 before node n-1 this way because Alice is left to Bob and he moves right first).

Let's see the following situation:

...B000A... and they both move right. The distance between them is 4 and they move right, after the next meetup, it will be 3 and the direction will move left etc. At the end the distance between them will be 1 (***...AB...***) and they will be moving left. In this situation they still will not be removing nodes one by one because Alice is moving away from Bob, so they will need one more meetup. So, when at the start Alice is moving away from Bob, they need one more meetup to start removing nodes one by one.

Knowing this, their positions in which they have the same direction for the first time thus the distance between them and the direction they are moving to, we can determine what node will be the last one standing. If at the beginning Alice moves away from Bob, we can just increase the initial distance by 1. If the initial distance is divisible by 2, by the time it becomes 1 they will be moving in the opposite direction.

If the initial distance is 4, they are moving left and Alice is not moving away from Bob, when the distance becomes 1, they will be moving right and thus the last node standing will be node 0 (like in the first example above). If the initial distance is 4, they are moving right and Alice is moving away from Bob, we can just say that the initial distance between them is 5. When the distance becomes 1, they will be moving right, so also in this case node 0 is the last one to stand (like in the second example above).

Problem G : Weights

Premier League and Rising Stars

Author:
Pavle Janevski

Implementation and analysis:
Pavle Janevski
Ivan Jevtić

Statement:

You are given an array A of length N weights of masses $A_1, A_2 \ldots A_N$. No two weights have the same mass. You can put every weight on one side of the balance (left or right). You don't have to put weights in order $A_1,\ldots,A_N$. There is also a string S consisting of characters 'L' and 'R', meaning that after putting the i_{th} weight (not A_i, but i_{th} weight of your choice) the left or right side of the balance should be heavier. Find the order of putting the weights on the balance such that rules of string S are satisfied.

Input:

The first line contains one integer N - the number of weights
The second line contains n distinct integers $A_1, A_2,\ldots,A_N$ - weights given
The third line contains string S of length N consisting only of letters 'L' and 'R' - string determining which side of the balance should be heavier after putting the i_{th} weight of your choice

Output:

The output contains N lines. In every line, you should print one integer and one letter - integer representing the weight you are putting on the balance in that move and the letter representing the side of the balance where you are putting the weight. If there is no solution, print **-1**.

Constraints:

- $1 \leq N \leq 2 * 10^5$
- $1 \leq A_i \leq 10^9, 1 \leq i \leq N$

Example input 1:

5
3 8 2 13 7
LLRLL

Example output 1:

3 L
2 R
8 R
13 L
7 L

Explanation 1:

Explanation for the test case:

- after the 1st weight: 3 L (**left** side is heavier)
- after the 2nd weight: 2 R (**left** side is heavier)
- after the 3rd weight: 8 R (**right** side is heavier)
- after the 4th weight: 13 L (**left** side is heavier)
- after the 5th weight: 7 L (**left** side is heavier)

So, the rules given by string $\mathcal{S}$ are fulfilled and the order of putting the weights is correct.

Time and memory limit: 1s / 256 MB

Solution and analysis:

First, we sort the weights. Now there is one key observation in order to solve this task. If we look at a sorted subarray A[L:R] ($1 \leq l \leq r \leq N$), such that weights are distributed alternately (A_L on one side, A_{L+1} on opposite side…), heavier side of the balance will always be the one where we put the last weight, the heaviest one.

Proof: There are 2 options:

Option 1:
We have distributed equal number of weights on both sides and now on side 1 we have:
A_L, A_{L+2}, ..., A_{L+2*K} and on side 2 we have:
A_{L+1}, A_{L+3}, ... $A_{L+2*K+1}$.
On the side 2, every weight is heavier than the corresponding weight on the other side.
$A_{L+1} > A_L$, $A_{L+3} > A_{L+2}$, ... $A_{L+2*K+1} > A_{L+2*K+1} \rightarrow A_{L+2*i+1} > A_{L+2*i}$, $0 \leq i \leq K$.
It is easy to see why is side 2 heavier than side 1. So, the side that we put the last weight on is the heavier side.

Option 2:
We have put the last weight on side 2, and now we have A_L, A_{L+2}, ..., A_{L+2*K} on that side, and A_{L+1}, A_{L+3}, ... $A_{L+2*K-1}$ on side 1. If we eliminate A_L on side 2, we have the same number of weights on both sides. The pairs are:
$A_{L+1} - A_{L+2}$, $A_{L+3} - A_{L+4}$, ... $A_{L+2*K-1} - A_{L+2*K}$. From these pairs, we see that all the heavier weights from pairs belong to side 2. On top of that if we add the remaining weight A_{L00}, we can see the side that we put the last weight on is heavier.

Now after we have sorted the weights, we are going to find the number of switches from one letter to another in string S. So, one switch happens when we go from L to R or vice versa (example: RLR has 2 switches). Let's call the number of switches X. The first weight we are going to put on the balance, on the side represented with the first letter, is going to be the one on position $N - X$ in the sorted array. Now we keep two pointers, let's call them left and right pointer, and both of them are at the $(N - X)_{th}$ position at the beginning. We go through the string S, starting from the 2nd letter (because we have already put the first weight), and we have two cases. When we do not switch from one letter to another, we decrease the left pointer and put the weight on which the left pointer now points. The side we are putting the weight is opposite from the side we put the weight where the left pointer was previously pointing. We are doing the similar thing when we make the switch, the difference is just that we increase the right pointer and put it on the side different from the side we put the previous element the right pointer was pointing at.
Why does this work? We put the first weight on the side represented by the first letter of the string. Every time we make a switch, we move the right pointer and put it on the opposite side, that side is going to be exactly the side that should be heavier because we only move it when we make a switch, so X times in total. When we don't make a switch, we move the left pointer, and the balance doesn't change.

Note: It is always possible to find the answer, if you follow the algorithm above

Problem H: Shortest path

Premier League and Rising Stars

Author:
Ivan Jevtić

Implementation and analysis:
Ivan Jevtić
Petar Vasiljević

Statement:

You are given N points on an infinite plane with Cartesian coordinate system on it. N-1 points lay on one line, and one point isn't on that line. You are on point K at the start, and the goal is to visit every point. You can move between any two points in a straight line, and you can revisit points. What is the minimum length of the path?

Input:

The first line contains two integers: N - the number of points, and K - the number of the starting point.

Each of the next N lines contain two integers, A_i and B_i – coordinates of the i_{th} point.

Output:

The output contains one number – the shortest path to visit all given points starting from point K. The absolute difference between your output and the solution shouldn't exceed $\mathbf{10^{-6}}$.

Constraints:

- $3 \leq N \leq 2 * 10^5$
- $-10^6 \leq A_i, B_i \leq 10^6, 1 \leq i \leq N$

Example input 1:

```
5 2
0 0
-1 1
2 -2
0 1
-2 2
```

Example output 1:

```
7.478709
```

Explanation 1:

The shortest path consists of these moves:

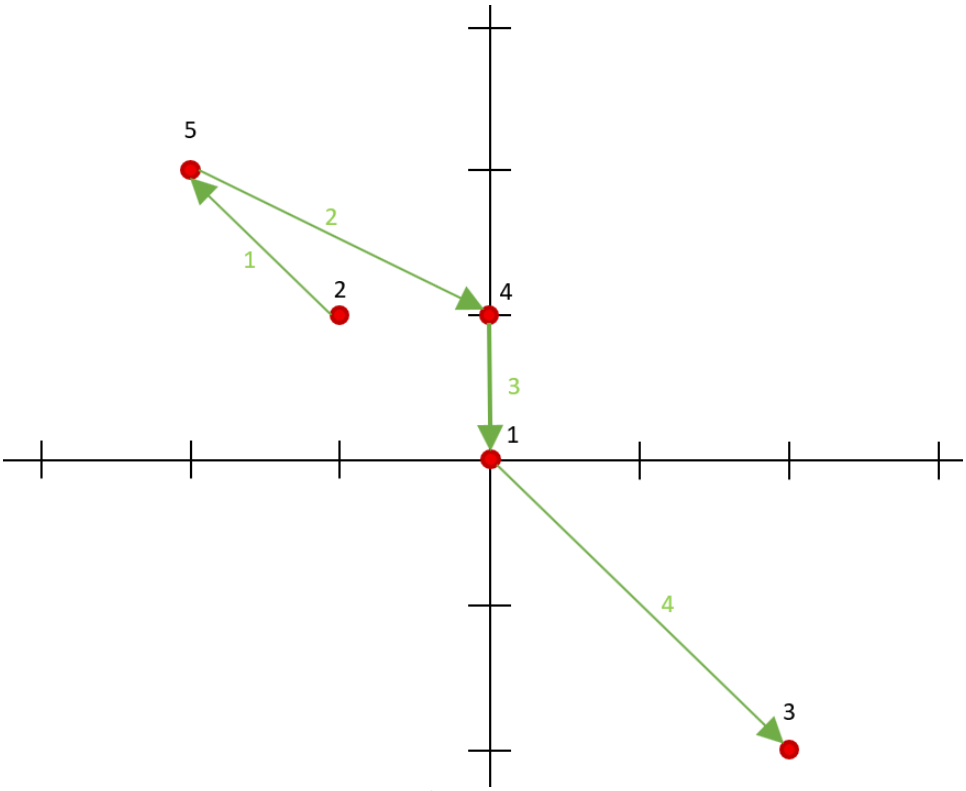
2 -> 5

5-> 4

4 -> 1

1 -> 3

There isn't any shorter path possible.



Time and memory limit: 1s / 256 MB

Solution and Analysis:

First, let us sort points by x coordinate. This way we can enumerate points in some order. Besides that, we also must find a point that is not on the given line. We can solve this with some easy math. One of the solutions is to take first 4 points from sorted order and calculate values of slopes between 1^{st} point (by order) and other 3. If there is a value that differs from the other 2, then that point is outside the line. Otherwise, just keep iterating over points and find the one such that a slope is different than the ones calculated before.

Now, let us enumerate points on the same line from 1 to $N-1$ and the point outside the line as N. Also, let's say that we have some indicators of leftmost and rightmost unvisited points, L and R. In our case, at the beginning, $L=1$ and $R=N-1$.

Observation: We are never going to alternate moves between two points such that none of them is one of L or R. This way we would never "shrink" our range so pathing would not become optimal. Therefore, optimal strategy is to reach L or R as soon as possible.

So, let us fix some point i such that $K <= i < R$ and say that in two passes we reach both L and i. There are two possibilities, one is to do movement $K -> i -> L$ and the other one is $K -> L -> i$. In both cases, next move is to reach point N that is not part of the line and then come back to point $i+1$. After that we should just go from $i+1$ to R. Therefore, solution for such i is:

$$min\begin{pmatrix} dist(K,i) + dist(i,1) + dist(1,N) + dist(N,i+1) + dist(i+1,N-1) \\ dist(K,1) + dist(1,i) + dist(i,N) + dist(N,i+1) + dist(i+1,N-1) \end{pmatrix}$$

Similar thing can be calculated in case that we pick such i where $L < i <= K$. In this case the solution is:

$$min\begin{pmatrix} dist(K,i) + dist(i,N-1) + dist(N-1,N) + dist(N,i-1) + dist(i-1,1) \\ dist(K,N-1) + dist(N-1,i) + dist(i,N) + dist(N,i-1) + dist(i-1,1) \end{pmatrix}$$

We can use the same formula for cases $i = L$ and $i = R$ but we must be careful of points $i + 1$ or $i - 1$ that don't exist in one of these formulas (just count it as a distance equal to 0).

One edge case is if starting point is the one outside of the line. Solution for this case is simply:

$$dist(1,N-1) + \min\big(dist(1,N), dist(N-1,N)\big)$$

Overall time complexity is $O(NlogN)$ since we're sorting points by x coordinate.

Problem I: Bob's Beautiful Array

Premier League and Rising Stars

Authors:

Nikola Pešić

Renea Mošo

Implementation and analysis:

Nikola Pešić

Renea Mošo

Statement:

Bob really likes playing with arrays of numbers. That's why for his birthday, his friends bought him an interesting machine – an array beautifier. The array beautifier takes an array A consisting of N integers, and it outputs a new array B of length N that it constructed based on the array given to it. The array beautifier constructs the new array in the following way: it takes two numbers **at different indices** from the original array and writes their sum to the end of the new array. It does this step N times - resulting in an output array of length N. During this process, the machine can take the same index multiple times in different steps. Bob was very excited about the gift that his friends have given him, so he put his favorite array in the machine. However, when the machine finished, Bob was not happy with the resulting array. He misses his favourite array very much and hopes to retrieve it. Given the array that the machine outputted, help Bob find an array that could be the original array that he put in the machine. Sometimes the machine makes mistakes, so it is possible that no appropriate input array exists for the array it has outputted. In such case, let Bob know that his array is forever lost.

Input:

The first line contains one positive integer N – the length of the array the machine outputted.

The second line contains N integers – the elements of the array the machine outputted.

Output:

If an appropriate input array exists, print "*YES, "* followed by the input array in the next line.

Otherwise, print "*NO. "*

Constraints:

- $2 \leq N \leq 10^3$
- $1 \leq B_i \leq 10^6$, *the array the machine outputted*
- $-10^9 \leq A_i \leq 10^9$, $Bob's\ original\ array$

Example input 1:

```
2
5 5
```

Example output 1:

```
YES
2 3
```

Example input 2:

```
3
1 2 3
```

Example output 2:

```
YES
0 1 2
```

Example input 3:

```
3
2 4 5
```

Example output 3:

```
NO
```

Example input 4:

```
4
1 3 5 7
```

Example output 4:

```
YES
6 -3 4 1
```

Time and memory limit: 1s / 256 MB

Solution and analysis:

If the array the machine outputted has any duplicates, we can just remove the duplicates, and replace them with zeros.

Now let us look at the elements in the original array as nodes in a graph, and an operation that the machine performs on two elements as an edge between those two elements. Now we have a graph with N nodes and N edges, notice that there must exist a cycle in this graph.

If we can find some elements in the array that machine outputted that we can form a cycle with, we can "attach" the remaining elements to some node in the cycle to generate an appropriate input array. Let's take for an example an output array "1 3 5 7 11 13, "notice that we can form a cycle with elements 1 3 5 7: input elements 0 3 4 1 are an example of 4 nodes that will form the cycle (0+3=3, 3+4=7, 4+1=5, 1+0=1). Now when we have the cycle, we can for example choose 1 as the node we will "attach" the remaining elements to, and get "0 3 4 1 10 12" as a valid input array. So, our problem is reduced to finding a cycle.

Now let us discuss when we can find a cycle. For a cycle of 3 elements "A B C" in the output array, lets mark one of the 3 elements in the input array as X, now we will have the two other elements be $A - X$, and $B - A + X$. When we look at the C branch that connects $B - A + X$ and X, we find that the following must hold: $C - B + A - X = X$, or $X = \frac{C-B+A}{2}$. This means that if we can find 3 numbers A, B and C such that their sum is divisible by 2, we can construct a solution. This will happen when we have 3 even numbers, or 2 odd and 1 even number.

Notice that for $N \geq 4$, if we have an even number in the output array, we will always be able to find an appropriate triple. For $N < 4$, if there are no duplicates, and we can not form a cycle of length 3 as described, the answer is "NO. "

Now we are left to deal with the case where we have no duplicates, and all the elements are odd. Notice that for every cycle of odd length, a similar rule as the one for a length of 3 must be satisfied (the sum of the numbers must be even), and that can never happen if all the numbers are odd.

So, we are now left with searching for a cycle of even length. Let us write out the equations for a cycle of length 4. Label the numbers in the array the machine outputted as "A B C D" and we will do the same thing we did for the length of 3. Start with X, the next element will be $A - X$, the next $B - A + X$ and the final will be $C - B + A - X$, now with the variable D, we will have the equation $D - C + B - A + X = X$, the X cancels out and we have $B + D = A + C$. So, for a cycle of length 4, we need two pairs that have the same sum. For a length 6 cycle we will need two sets of 3 numbers with the same sum. For 8, we will need two sets of 4 numbers with the same sum, and so on.

If we cannot find two sets of the same size with the same sum, the answer is "NO" since there is no way left to form a cycle.

Once we have two sets (of the same size) with the same sum, we can form a cycle by taking a number from the first set for the first edge, a number from the second for the second edge, a number from the first set for the third edge and continue alternating between the sets to form a valid cycle. As X canceled out in our equation, we can set the first node in our cycle to any value, for example 0. And once we

finish our cycle, we can just append the remaining elements from the array the machine outputted that were not in the cycle since we have a 0 in the array.

If we have N numbers in the output array, and we are looking for two sets of size S with the same sum, we will have $\binom{N}{S}$ different sets and $10^6 * S$ different values for the sum. If the number of sets is greater than the number of different values, we will always be able to find two different sets with the same sum (the pigeonhole principle).

Note: if the two sets we find have some numbers appearing in both sets, we can just remove those numbers from both sets and get two sets with the same sum of some smaller size.

Now we can form our solution:

Go through the sets of some size S in order and we are looking for two sets with the same sum. If we find two sets with the same sum, we can for our solution. If we fail to find two appropriate sets of size S, we need to try using some larger value of S (up to $\left\lfloor \frac{N}{2} \right\rfloor$). For any $N \geq 26$, we are guaranteed to find a solution, so in those cases, we need to select the smallest S for which it is guaranteed we find a solution and go through the sets of size S in some order, memorizing the sums we already have. If we use a hash table for this memorization, the complexity of this approach will be $O(S^2 * 10^6)$. And for $N < 26$, we will at most go through all the subsets of size less than or equal to $\left\lfloor \frac{N}{2} \right\rfloor$, with the complexity $O(2^N * S)$.

Problem J: Mars

Premier League and Rising Stars

Author:

Igor Pavlović

Implementation and analysis:

Igor Pavlović

Pavle Janevski

Statement:

In the year 2420 humans have finally built a colony on Mars thanks to the work of Elon Tusk. There are $10^9 + 7$ cities arranged in a circle in this colony and none of them are connected yet. Elon Tusk wants to connect some of those cities using only roads of the same size in order to lower the production cost of those roads. Because of that he gave a list on N cites where some cites can appear more than once and Q queries that you need to answer. For the query you need to determine if it is possible to connect all the cities from L_i to R_i on that list using only roads of length D_i.

Input:

The first line contains two integers N and Q — the length of the array of cities and the number of queries you need to answer.

The second line contains N integers representing the array of cites.

Next Q lines contain three integers L, R and D — the range of cities that needs to be connected and the length of the road that you can use.

Output:

The output contains Q lines. If it is possible to connect all the cities from the i^{th} query can be connected with roads of length D_i the i^{th} line should contain the word "Yes," otherwise it should contain the word "No."

Constraints:

- $1 \leq N, Q \leq 2 * 10^5$
- $1 \leq L_i, R_i \leq N$
- $0 \leq D_i \leq 10^9 + 6$

Example input 1:

```
9 8
17 0 12 6 10 8 2 4 5
2 3 12
2 3 6
2 4 6
4 6 2
2 8 2
1 2 17
1 8 2
```

```
9 9 14
```

Example output 1:

```
Yes

No

Yes

Yes

Yes

Yes

No

Yes
```

Explanation 1:

In the 5^{th} query we can connect cities in this order 0-2-4-6-8-10-12 this way distance between any two connected cities is 2.

Example input 2:

```
4 1
7 21 14 0
1 4 1000000000
```

Example output 2:

```
Yes
```

Explanation 2:

We can connect cities in this order 21-14-7-0 this way distance between any two connected cities is 10^9 module $10^9 + 7$.

Time and memory limit: 1s / 256 MB

Solution and analysis:

It is not hard to see that behind the statement there is actually the problem that should sounds less complicated: Given L_i, R_i and D_i, question is can we arrange elements **a[L_i, R_i]** into arithmetic progression modulo $10^9 + 7$ such that difference between consecutive elements is D_i.

This can be done using hashing by checking if the sum of **k-th** powers of elements **a[L_i, R_i]** is equal to the sum of **k-th** of the right arithmetic progression modulo $10^9 + 7$. Ideally this number **k** should be larger but this should also work for smaller values of **k.**

First, we want to know which element is the first in the arithmetic progression. Knowing the length of the progression and difference between consecutive elements, we can calculate the first element of the progression. If we have ***n*** elements in the progression ($\boldsymbol{n = Ri - Li + 1}$) and the difference is ***d*** (***d = D_i***), sum of the elements in the progression is

$\boldsymbol{S = \frac{n}{2} * (2 * A_0 + (n - 1) * d)}$. Out of this equation we can find A_0 – first element of the sequence.

Finding the hash of **a[L_i, R_i]** can be done for any number **k** by using the prefix sums of **k-th** powers for the original array. The hash of an arithmetic progression can be written as $\sum_{i=0}^{n-1}(A_0 + i * d)^k$, or ${A_0}^k * \sum_{i=0}^{n-1}(1 + i * x)^k = {A_0}^k * P(x)$, where $x = \frac{d}{A_0}$ modulo $10^9 + 7$. $P(x)$ can be written as $P(x) = \sum_{i=0}^{k}\binom{k}{i} * S_{n,i} * x^i$, where $S_{n,i}$ is the sum of ***i-th*** powers of all non-negative integers less than ***n***. We can use dynamic programming to find the coefficients of this polynomial for every $n \leq N$ in $O(N * k)$.

After we find the coefficients of these polynomials, we can use it to find the hash of any arithmetic progression in $O(k)$, which will give us a total complexity of $O((N + Q) * k)$. In order to improve the accuracy of this algorith we can choose two or more different walues of ***k***, for example $k1 = 51$ and $k2 = 52$.

Problem K: Two Arrays

Premier League only

Author:

Tadija Šebez

Implementation and analysis:

Tadija Šebez

David Milićević

Statement:

You are given two integer arrays of length N, $A1$ and $A2$. You are also given Q queries of 4 types:

- $\mathbf{1\ k\ l\ r\ x}$: set $Ak_i := min(Ak_i, x)$ for each $l \leq\ i \leq\ r$.
- $\mathbf{2\ k\ l\ r\ x}$: set $Ak_i := max(Ak_i, x)$ for each $l \leq\ i \leq\ r$.
- $\mathbf{3\ k\ l\ r\ x}$: set $Ak_i := Ak_i + x$ for each $l \leq\ i \leq\ r$.
- $\mathbf{4\ l\ r}$: find the $\left(\Sigma_{i=l}^{r} F(A1_i + A2_i)\right)\ \%\ (10^9 + 7)$ where $F(k)$ is the k^{th} Fibonacci number $(F(0) = 0, F(1) = 1, F(k) = F(k-1) + F(k-2))$, and $x\ \%\ y$ denotes the remainder of the division of x by y.

You should process these queries and answer each query of the fourth type.

Input:

The first line contains two integers N and Q.

The second line contains N integers, array $A1_1, A1_2, \ldots\ A1_N$.

The third line contains N integers, array $A2_1, A2_2, \ldots\ A2_N$.

The next Q lines describe the queries. Each line contains 5 or 3 integers, where the first integer denotes the type of the query.

Output:

Print the answer to each query of the fourth type, in separate lines.

Constraints:

- $1 \leq\ N, Q \leq\ 5\ \times\ 10^4$
- $0 \leq\ A1_i,\ A2_i \leq\ 10^6$
- $k \in\ \{\ 1, 2\ \}$
- $1 \leq\ l \leq\ r \leq\ N$
- ***For queries of type 1 and 2***, $0 \leq\ x\ \leq 10^9$

- ***For queries of type 3***, $-10^6 \leq x \leq 10^6$
- ***It is guaranteed that after every query each number in arrays*** $A1$ ***and*** $A2$ ***will be nonnegative***.

Example input 1:

```
3 4
1 0 2
2 1 0
4 1 3
3 2 2 2 3
1 1 1 3 0
4 1 3
```

Example output 1:

```
4
4
```

Explanation 1:

- The answer for the first query is $F(1 + 2) + F(0 + 1) + F(2 + 0) = F(3) + F(1) + F(2) = 2 + 1 + 1 = 4$.
- After the second query, the array $A2$ changes to $[2, 4, 0]$.
- After the third query, the array $A1$ changes to $[0, 0, 0]$.
- The answer for the fourth query is $F(0 + 2) + F(0 + 4) + F(0 + 0) = F(2) + F(4) + F(0) = 1 + 3 + 0 = 4$.

Example input 2:

```
5 4
1 3 5 3 2
4 2 1 3 3
4 1 3
4 2 5
2 1 2 4 6
4 2 4
```

Example output 2:

```
18
26
68
```

Explanation 2:

- The answer for the first query is $F(1 + 4) + F(3 + 2) + F(5 + 1) = F(5) + F(5) + F(6) = 5 + 5 + 8 = 18$.
- The answer for the second query is $F(3 + 2) + F(5 + 1) + F(3 + 3) + F(2 + 3) = F(5) + F(6) + F(6) + F(5) = 5 + 8 + 8 + 5 = 26$.
- After the third query, the array $A1$ changes to $[1, 6, 6, 6, 2]$.

- The answer for the fourth query is $F(6 + 2) + F(6 + 1) + F(6 + 3) = F(8) + F(7) + F(9) = 21 + 13 + 34 = 68$.

Time and memory limit: 3s / 256MB

Solution and analysis:

We will use Segment Tree Beats (STB) to solve this problem. Basic idea and more details about STB can be found at https://codeforces.com/blog/entry/57319 (where STB was firstly introduced in English) and https://codeforces.com/blog/entry/90460 (where you have nice video explanation).

First part of solution consists of efficient calculation of N-th Fibonacci number. For this purpose we can use matrix $F = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. To get the value of N-th Fibonnaci number, we simply calculate F^N and take the value at $F[1][0]$ or $F[0][1]$. To speed-up the calculation further, we can take some fixed number H (10^6 for example). Then we can calculate F^i for each i up to N and store that values in some array y. Also, in another array x we can store F^{iH}. This way, we can caclulate N-th Fibonacci number by caclulating matrix $x[N/H] * y[N\%H]$.

To be able to use STB, fore each node of segment tree we need to store few things – nine 2x2 matrices, six lazy tags, and some additional info.

Additional info consists of maximum value, second maximum value, minimum value and second minimum value on the segment. Those are used as in standard STBs (examples in the links above).

Lazy tags are stored for both arrays and they correspond to the first three types of query. We store $lazyMin$ – minimal value on the segment, $lazyMax$ – maximal value on the segment and $lazyAdd$ – value that needs to be added to the segment.

Let $mx_1 = max_{i \in segment}(A_1[i])$, $mx_2 = max_{i \in segment}(A_2[i])$, $mn_1 = min_{i \in segment}(A_1[i])$ and $mn_2 = min_{i \in segment}(A_2[i])$. Then, nine matrices that are being stored are:

1. $\sum_{\{i \in segment, A_1[i] \notin \{mn_1, mx_1\}, A_2[i] \notin \{mn_2, mx_2\}\}} (F^{A_1[i]+A_2[i]})$
2. $\sum_{\{i \in segment, A_1[i] = mx_1, A_2[i] \notin \{mn_2, mx_2\}\}} (F^{A_1[i]+A_2[i]})$
3. $\sum_{\{i \in segment, A_1[i] \notin \{mn_1, mx_1\}, A_2[i] = mx_2\}} (F^{A_1[i]+A_2[i]})$
4. $\sum_{\{i \in segment, A_1[i] = mx_1, A_2[i] = mx_2\}} (F^{A_1[i]+A_2[i]})$
5. $\sum_{\{i \in segment, A_1[i] = mn_1, A_2[i] \notin \{mn_2, mx_2\}\}} \left(F^{A_1[i]+A_2[i]}\right)$
6. $\sum_{\{i \in segment, A_1[i] \notin \{mn_1, mx_1\}, A_2[i] = mn_2\}} (F^{A_1[i]+A_2[i]})$
7. $\sum_{\{i \in segment, A_1[i] = mn_1, A_2[i] = mn_2\}} (F^{A_1[i]+A_2[i]})$
8. $\sum_{\{i \in segment, A_1[i] = mn_1, A_2[i] = mx_2\}} (F^{A_1[i]+A_2[i]})$
9. $\sum_{\{i \in segment, A_1[i] = mx_1, A_2[i] = mn_2\}} (F^{A_1[i]+A_2[i]})$

Those nine matrices, represent all possible combinations where each of elements from A_1 and A_2 is either minimal value on the segment, maximal value, or the value between them. When summed, those nine matrices represent a result of the query 4 over the segment they belong to.

Now we just need to be cautious with special cases, like when there is only one value in the segment or only two different values in a segment, etc.

The only thing left to do is to come up with $break\_condition$ and $tag\_condition$ methods. Method $break\_condition$, together with standard interval checks, tells when to stop propagating the value down the tree during update operations. It needs to be true for both arrays in order to stop

propagation. In our case, it's true only if $lazyAdd = 0$ and $mx < lazyMax$ and $mn > lazyMax$, where mx and mn represent minimum and maximum value on the segment. Method $tag\_condition$, together with standard interval checks, tells when to update lazy tags and stop further propagation. It also needs to be true for both arrays in order to update lazy tags. In our case, it's true when $lazyAdd \neq 0$ or, $lazyMax > smx$ and $lazyMin < smn$, where smx and smn represend second maximum and second minimum value on the segment, respectivelly (you can check links above to see in more details why smx and smn are used this way).

To count the complexity, we will divide our solution into three parts – preprocessing, building segment tree and processing queries. As usual, first two parts are not significant in total complexity. Prepocessing for Fibonacci numbers calculation (as explained above) is $O(H)$, which is constant, so we don't count it in total complexity. For the second part, complexity is $O(kN)$, where k is a small constant factor due to matrix multiplication in order to calculate Fibonacci matrices in leaf nodes of segment tree.

Third part, query processing, has complexity of $O(lQlog^2(N))$. Proof for $O(Qlog^2(N))$ part can be found on the first blog post referenced at the beginning of this solution. It's the same task, the only difference being query 4 which is simpler – it's just basic sum calculation. Here, we have a constant factor l. This factor is hard to calculate since it comes from multiple things being done while propagating values down/up the tree. For example, someone could have used standard fast matrix exponentiation instead of exponentiation explained above – this would have added another log factor to the complexity which wouldn't be enough to pass time limits.

Thus, complexity is $O\big(N + Qlog^2(N)\big)$, but implementation needs to be careful in order to reduce constant factors as much as possible.

Problem L: Bubble Popping

Premier League only

Author:

Tadija Šebez

Implementation and analysis:

Tadija Šebez

Aleksandar Damjanović

Statement:

There are N bubbles in a coordinate plane. Bubbles are so tiny that it can be assumed that each bubble is a point (X_i, Y_i).

Q Bubble Cup finalists plan to play with the bubbles. Each finalist would link to use infinitely long Bubble Cup stick to pop some bubbles. The i^{th} finalist would like to place the stick in the direction of vector (dx_i, dy_i), and plays the following game until K_i bubbles are popped. The game starts with finalist placing the stick in the direction of vector (dx_i, dy_i), and sweeping it from the infinity to the left until it hits some bubble, which is immediately popped. It is guaranteed that only one bubble will be hit in this step. After that the finalist starts rotating the stick in the counter clockwise direction with the center of rotation in point where the previous bubble was popped. When the next bubble is hit, it is immediately popped and becomes the new center of rotation. The process continues until K_i bubbles have been popped. It is guarnteed that the stick won't hit two bubbles simultaneously in this process.

For each finalist, find which bubble would be popped the last. Note that each game starts with the configuration of all N bubbles, so the games don't depend on the previous games.

Input:

The first line contains one integer N — the number of bubbles.

Each of the next N lines contain two integers. The i^{th} line contains integers X_i and Y_i — the coordinates of the i^{th} bubble.

The next line contains one integer Q — the number of finalists willing to play with the bubbles.

Each of the next Q lines contains 3 integers. The i^{th} line contains integers dx_i, dy_i and K_i.

Output:

For each of the Q finalists, print the index of the bubble which would be popped last, in the separate line.

Constraints:

- $1 \leq N, Q \leq 10^5$
- $-10^9 \leq X_i, \mathrm{Y_i} \leq 10^9$

- $(X_i, Y_i) \neq (X_j, Y_j), for\ i \neq j.$
- $-10^9 \leq dx_i, dy_i \leq 10^9$
- $1 \leq K_i \leq N$

Example input 1:

```
4
0 0
1 0
0 1
1 1
2
1 -1 3
-1 1 4
```

Example output 1:

```
4
2
```

Explanation 1:

There are two finalists willing to play with the bubbles. If the first finalist plays with the bubbles, then the bubbles at coordinates (0, 0), (1, 0) and (1, 1) would be popped in that order. Their indexes are 1, 2 and 4, so the answer is 4. If the second finalist plays with the bubbles, then the bubbles at coordinates (1, 1), (0, 1), (0, 0) and (1, 0) would be popped in that order, so the answer is 2.

Example input 2:

```
4
1 1
2 2
7 1
1 7
3
2 2 1
1 -5 4
-6 5 3
```

Example output 2:

```
3
2
3
```

Explanation 2:

Time and memory limit: 3s / 256 MB

Solution and Analysis:

Solution idea

Let us name first bubble that is popped $B_{1,1}$. It is immediately evident that after first bubble is popped, we will keep popping bubbles in counter clockwise largest convex hull order starting with $B_{1,1}$ until we finish with the last point of the largest Conex hull. Let's say this largest convex hull contains m_1 points. If $K_i \leq m_1$ we can immediately say that last popped bubble is B_{1,K_i} . Now let's consider option when $K_i > m_1$. In that case after B_{1,m_1} bubble is popped we must find the next point $B_{2,1}$ from which we will start again be popping bubbles in the largest convex hull order $B_{2,1}$, $B_{2,2}$, ... B_{2,m_2}. Therefore, we will have nested set of convex hulls with sizes $m_1, m_2, m_3 \ldots$. We will continue this process until $m_1 + m_2 + .. + m_k + x = K_i$. Therefore K_i -th point that will be popped is $B_{k+1,x}$

Algorithm analysis

Repeating standard $O(n\ log\ n)$ convex hull algorithm on different layers would result in $O(N^2\ log\ N)$ time to create these layers which is too slow. Therefore, we must use decremental convex hull algorithm that can find next convex hull point in cases when points are deleted (bubbles are popped). This algorithm is described in paper:

M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166-204, 1981.

Using this approach our overall complexity for calculating convex layers would be $O(N\ log^2\ N)$. Once we finish creating convex hull layers, we need to find transition points between two consecutive layers (next point in the layer below assuming current point is the last point in a layer). This can be done using binary search in $O(N\ log\ N)$ since the layers already have order. Finally, as we are answering Q queries we need to quickly make transitions between convex hull layers. For this we can use binary lifting technique. After all this pre-processing for each query we can in $O(log\ N)$ time find first point in the layer where solution is located.

Overall algorithm complexity of this solution is $O(N \log^2 N\ +\ Q \log N)$.

Problem M: Desert

Premier League only

Author:

Tadija Šebez

Implementation and analysis:

Tadija Šebez

Kosta Grujčić

Statement:

You are given an undirected graph of N nodes and M edges, $E_1, E_2, \ldots E_M$.

A connected graph is a cactus if each of its edges belong to at most one simple cycle. A graph is a desert if each of its connected components is a cactus.

Find the number of pairs (L, R), $(1 \leq L \leq R \leq M)$ such that if we delete all the edges except for $E_L, E_{L+1}, \ldots E_R$, the graph is a desert.

Input:

The first line contains two integers N and M. Each of the next M lines contain two integers. The i^{th} line describes the i^{th} edge. It contains integers U_i and V_i, the nodes connected by the i^{th} edge $(E_i = (U_i, V_i))$. It is guaranteed that $U_i \neq V_i$.

Output:

The output contains one integer number – the answer.

Constraints:

- $2 \leq N \leq 2.5 \times 10^5$
- $1 \leq M \leq 5 \times 10^5$
- $1 \leq U_i, V_i \leq N$
- $U_i \neq V_i$

Example input 1:

```
5 6
1 2
2 3
3 4
4 5
5 1
2 4
```

Example output 1:

```
20
```

Example input 2:

```
2 3
1 2
1 2
1 2
```

Example output 2:

```
5
```

Explanation 2:

- Graphs for pairs $(1, 1)$, $(2, 2)$ and $(3, 3)$ are deserts because they don't have any cycles.
- Graphs for pairs $(1, 2)$ and $(2, 3)$ have one cycle of length 2 so they are deserts.

Time and memory limit: 2.0s / 256 MB

Solution and Analysis:

First, note that if a graph is a desert, then it stays a desert if we remove some edges from it. Similarly, if a graph is not a desert, then it won't be a desert if we add more edges to it. It follows that we can use two pointers technique to find the maximum R , for each L , such that the graph is a desert for the pair (L, R).

To pass the time limit, we obviously need some fast data structure that supports adding and removing edges, and after each change tells us if the graph is still a desert. There is a similar data structure for trees called link-cut trees, or LCT for short. It turns out that we can use LCT in some clever way, to fulfil its needs.

We will maintain a spanning tree of each connected component of the desert. Note that exactly one edge from each cycle will not be in the spanning trees. It would be nice to somehow maintain which edges are on a cycle and which are not. Unfortunately, LCT cannot store and maintain information about edges. To overcome this issue, we will split each edge in half and insert a node in between. This way we can store the information about the edges in these new nodes.

Now, let us see what happens when we add an edge to the desert. If its endpoints are not connected in the spanning forest, then we can add it and connect two cactuses. Obviously, in this case we will not produce any new cycles thus the graph stays a desert. On the other hand, if its endpoints are already connected, we should look at the edges on the path between these endpoints. If there is an edge that is on a cycle the graph is no more a desert, so we cannot add a new edge. Otherwise, we form a new cycle. One of the edges on the cycle will not be in the spanning tree, let it be the edge that will be deleted before all the other edges, i.e. the edge with the smallest index. Remove that edge from the tree and add the new edge. Then mark all the other edges as edges on a cycle. Obviously, we cannot just iterate over all the edges. We use the lazy propagation technique instead.

The only thing left is to support edge removals. There are two cases. If the edge is on a cycle, it will not be in a spanning tree because of the way we choose which edges are in the spanning tree. Thus, we can just mark the rest of the edges on cycle as not being on a cycle anymore, again using lazy propagation. If the edge is not on a cycle, we can simply remove it from the spanning tree.

Overall time complexity of this solution is $O(M \log N)$.

There is also an extension of LCT, called link-cut cactus (or LCC), which supports exactly these operations we need, and many more. Using LCC instead of LCT also passes the time limit, but the time complexity is slightly worse – $O(M \log^2 N)$.